Standard glossary of terms used in Software Testing

**Version 2.3 (dd. March 28th, 2014)**

**Produced by the ‘Glossary Working Party’**

**International Software Testing Qualifications Board**

**1.1**

**bug:** See *defect*.

**fault:** See *defect*.

**defect:** A flaw in a component or system that can cause the component or system to fail to

perform its required function, e.g. an incorrect statement or data definition. A defect, if

encountered during execution, may cause a failure of the component or system.

**mistake:** See *error*.

**error:** A human action that produces an incorrect result.

**failure:** Deviation of the component or system from its expected delivery, service or result.

**quality:** The degree to which a component, system or process meets specified im

and/or user/customer needs and expectations.

**risk:** A factor that could result in future negative consequences; usually expressed as impact and likelihood.

**1.2**

**debugging:** The process of finding, analyzing and removing the causes of failures in

software.

**requirement:** A condition or capability needed by a user to solve a problem or achieve an

objective that must be met or possessed by a system or system component to satisfy a

contract, standard, specification, or other formally imposed document.

**review:** An evaluation of a product or project status to ascertain discrepancies from planned

results and to recommend improvements. Examples include management review, informal

review, technical review, inspection, and walkthrough.

**test case:** A set of input values, execution preconditions, expected results and execution postconditions, developed for a particular objective or test condition, such as to exercise a

particular program path or to verify compliance with a specific requirement.

**testing:** The process consisting of all lifecycle activities, both static and dynamic, concerned

with planning, preparation and evaluation of software products and related work products

to determine that they satisfy specified requirements, to demonstrate that they are fit for

purpose and to detect defects.

**test objective:** A reason or purpose for designing and executing a test.

**1.3**

**exhaustive testing:** A test approach in which the test suite comprises all combinations of

input values and preconditions.

**1.4**

**confirmation testing:** Testing that runs test cases that failed the last time they were run, in

order to verify the success of corrective actions.

**re-testing:** See *confirmation testing.*

**exit criteria:** The set of generic and specific conditions, agreed upon with the stakeholders

for permitting a process to be officially completed. The purpose of exit criteria is to

prevent a task from being considered completed when there are still outstanding parts of

the task which have not been finished. Exit criteria are used to report against and to plan

when to stop testing.

**incident:** Any event occurring that requires investigation.

**regression testing:** Testing of a previously tested program following modification to ensure

that defects have not been introduced or uncovered in unchanged areas of the software, as a

result of the changes made. It is performed when the software or its environment is

changed.

**test basis:** All documents from which the requirements of a component or system can be

inferred. The documentation on which the test cases are based. If a document can be

amended only by way of formal amendment procedure, then the test basis is called a frozen

test basis.

**test condition:** An item or event of a component or system that could be verified by one or

more test cases, e.g. a function, transaction, feature, quality attribute, or structural element.

**test coverage:** See *coverage*.

**coverage:** The degree, expressed as a percentage, to which a specified coverage item has been

exercised by a test suite.

**test data:** Data that exists (for example, in a database) before a test is executed, and that

affects or is affected by the component or system under test.

**test execution:** The process of running a test on the component or system under test,

producing actual result(s).

**test log:** A chronological record of relevant details about the execution of tests.

**test plan:** A document describing the scope, approach, resources and schedule of intended

test activities. It identifies amongst others test items, the features to be tested, the testing

tasks, who will do each task, degree of tester independence, the test environment, the test

design techniques and entry and exit criteria to be used, and the rationale for their choice,

and any risks requiring contingency planning. It is a record of the test planning process.

**test procedure:** See *test procedure specification*.

**test procedure specification:** A document specifying a sequence of actions for the execution

of a test. Also known as test script or manual test script. See also *test*

*specification.*

**test specification:** A document that consists of a test design specification, test case

specification and/or test procedure specification.

**test policy:** A high level document describing the principles, approach and major objectives

of the organization regarding testing.

**test suite:** A set of several test cases for a component or system under test, where the post

condition of one test is often used as the precondition for the next one.

**test summary report:** A document summarizing testing activities and results. It also contains

an evaluation of the corresponding test items against exit criteria.

**testware:** Artifacts produced during the test process required to plan, design, and execute

tests, such as documentation, scripts, inputs, expected results, set-up and clear-up

procedures, files, databases, environment, and any additional software or utilities used in

testing.

**1.5**

**error guessing:** A test design technique where the experience of the tester is used to

anticipate what defects might be present in the component or system under test as a result

of errors made, and to design tests specifically to expose them.

**independence of testing:** Separation of responsibilities, which encourages the

ATM accomplishment of objective testing.

**2.1**

**COTS:** Acronym for Commercial Off-The-Shelf software. See *off-the-shelf software.*

**off-the-shelf software:** A software product that is developed for the general market, i.e. for a

large number of customers, and that is delivered to many customers in identical format.

**iterative development model:** A development lifecycle where a project is broken into a

usually large number of iterations. An iteration is a complete development loop resulting in

a release (internal or external) of an executable product, a subset of the final product under

development, which grows from iteration to iteration to become the final product.

**validation:** Confirmation by examination and through provision of objective evidence that

the requirements for a specific intended use or application have been fulfilled.

**verification:** Confirmation by examination and through provision of objective evidence that

specified requirements have been fulfilled.

**V-model:** A framework to describe the software development lifecycle activities from

requirements specification to maintenance. The V-model illustrates how testing activities

can be integrated into each phase of the software development lifecycle.

**2.2**

**alpha testing:** Simulated or actual operational testing by potential users/customers or an

independent test team at the developers’ site, but outside the development organization.

Alpha testing is often employed for off-the-shelf software as a form of internal acceptance

testing.

**beta testing:** Operational testing by potential and/or existing users/customers at an external

site not otherwise involved with the developers, to determine whether or not a component

or system satisfies the user/customer needs and fits within the business processes. Beta

testing is often employed as a form of external acceptance testing for off-the-shelf software

in order to acquire feedback from the market.

**component testing:** The testing of individual software components.

**driver:** A software component or test tool that replaces a component that takes care of the

control and/or the calling of a component or system.

**field testing:** See *beta testing*.

**functional requirement:** A requirement that specifies a function that a component or system

must perform.

**integration:** The process of combining components or systems into larger assemblies.

**integration testing:** Testing performed to expose defects in the interfaces and in the

interactions between integrated components or systems. See also *component integration*

*testing, system integration testing*.

**non-functional requirement:** A requirement that does not relate to functionality, but to

attributes such as reliability, efficiency, usability, maintainability and portability.

**robustness testing:** Testing to determine the robustness of the software product.

**stub:** A skeletal or special-purpose implementation of a software component, used to develop

or test a component that calls or is otherwise dependent on it. It replaces a called

component.

**system testing:** The process of testing an integrated system to verify that it meets specified

requirements.

**test environment:** An environment containing hardware, instrumentation, simulators,

software tools, and other support elements needed to conduct a test.

**test level:** A group of test activities that are organized and managed together. A test level is

linked to the responsibilities in a project. Examples of test levels are component test,

integration test, system test and acceptance test.

**test-driven development:** A way of developing software where the test cases are

developed, and often automated, before the software is developed to run those test cases.

**user acceptance testing:** See *acceptance testing*.

**acceptance testing:** Formal testing with respect to user needs, requirements, and business

processes conducted to determine whether or not a system satisfies the acceptance criteria

and to enable the user, customers or other authorized entity to determine whether or not to

accept the system.

**2.3**

**black box testing:** Testing, either functional or non-functional, without reference to the

internal structure of the component or system.

**code coverage:** An analysis method that determines which parts of the software have been

executed (covered) by the test suite and which parts have not been executed, e.g. statement

coverage, decision coverage or condition coverage.

**functional testing:** Testing based on an analysis of the specification of the functionality of a

component or system. See also *black box testing*.

**interoperability testing:** The process of testing to determine the interoperability of a

ATA software product. See also *functionality testing*.

**load testing:** A type of performance testing conducted to evaluate the behavior of a

component or system with increasing load, e.g. numbers of parallel users and/or numbers

of transactions, to determine what load can be handled by the component or system. See

also *performance testing, stress testing*.

**maintainability testing:** The process of testing to determine the maintainability of a software

product.

**performance testing:** The process of testing to determine the performance of a software

F-AT product. See also *efficiency testing*.

**efficiency testing:** The process of testing to determine the efficiency of a software product.

**portability testing:** The process of testing to determine the portability of a software product.

**reliability testing:** The process of testing to determine the reliability of a software product.

**security testing:** Testing to determine the security of the software product. See also

*functionality testing.*

**stress testing:** A type of performance testing conducted to evaluate a system or component at

or beyond the limits of its anticipated or specified workloads, or with reduced availability

of resources such as access to memory or servers. See also *performance*

*testing, load testing*.

**structural testing:** See *white-box testing*.

**white-box testing:** Testing based on an analysis of the internal structure of the component or

system.

**usability testing:** Testing to determine the extent to which the software product is

understood, easy to learn, easy to operate and attractive to the users under specified

conditions.

**2.4**

**impact analysis:** The assessment of change to the layers of development documentation, test

documentation and components, in order to implement a given change to specified

requirements.

**maintenance testing:** Testing the changes to an operational system or the impact of a

changed environment to an operational system.

**3.1**

**dynamic testing:** Testing that involves the execution of the software of a component or

**system.**

**static testing:** Testing of a software development artifact, e.g., requirements, design or code,

without execution of these artifacts, e.g., reviews or static analysis.

**3.2**

**entry criteria:** The set of generic and specific conditions for permitting a process to go

forward with a defined task, e.g. test phase. The purpose of entry criteria is to prevent a

task from starting which would entail more (wasted) effort compared to the effort needed

to remove the failed entry criteria.

**formal review:** A review characterized by documented procedures and requirements, e.g.

inspection.

**informal review:** A review not based on a formal (documented) procedure.

**inspection:** A type of peer review that relies on visual examination of documents to detect

defects, e.g. violations of development standards and non-conformance to higher level

documentation. The most formal review technique and therefore always based on a

documented procedure.

**metric:** A measurement scale and the method used for measurement.

**moderator:** The leader and main person responsible for an inspection or other review

process.

**peer review:** A review of a software work product by colleagues of the producer of the

product for the purpose of identifying defects and improvements. Examples are inspection,

technical review and walkthrough.

**reviewer:** The person involved in the review that identifies and describes anomalies in the

product or project under review. Reviewers can be chosen to represent different viewpoints

and roles in the review process.

**scribe:** The person who records each defect mentioned and any suggestions for process

improvement during a review meeting, on a logging form. The scribe should ensure that

the logging form is readable and understandable.

**technical review:** A peer group discussion activity that focuses on achieving consensus on

the technical approach to be taken.

**walkthrough:** A step-by-step presentation by the author of a document in order to gather

information and to establish a common understanding of its content. See also *peer review*.

**peer review:** A review of a software work product by colleagues of the producer of the

product for the purpose of identifying defects and improvements. Examples are inspection,

technical review and walkthrough.

**3.3**

**compiler:** A software tool that translates programs expressed in a high order language into

their machine language equivalents.

**complexity:** The degree to which a component or system has a design and/or internal

structure that is difficult to understand, maintain and verify. See also *cyclomatic*

*complexity.*

**cyclomatic complexity:** The maximum number of linear, independent paths through a

program. Cyclomatic complexity may be computed as: L – N + 2P, where

- L = the number of edges/links in a graph

- N = the number of nodes in a graph

- P = the number of disconnected parts of the graph (e.g. a called graph or subroutine)

**control flow:** A sequence of events (paths) in the execution through a component or system.

**data flow:** An abstract representation of the sequence and possible changes of the state of

data objects, where the state of an object is any of: creation, usage, or destruction.

**static analysis:** Analysis of software development artifacts, e.g. requirements or code, carried

out without execution of these software development artifacts. Static analysis is usually

carried out by means of a supporting tool.

**4.1**

**test case specification:** A document specifying a set of test cases (objective, inputs, test

actions, expected results, and execution preconditions) for a test item.

See also *test specification.*

**test design:** (1) See *test design specification*.

(2) The process of transforming general test objectives into tangible test conditions and

test cases.

**test design specification:** A document specifying the test conditions (coverage items) for a

test item, the detailed test approach and identifying the associated high level test cases.

See also *test specification.*

**test execution schedule:** A scheme for the execution of test procedures. Note: The test

procedures are included in the test execution schedule in their context and in the order in

which they are to be executed.

**test procedure specification:** A document specifying a sequence of actions for the execution

ATM of a test. Also known as test script or manual test script.

**test script:** Commonly used to refer to a test procedure specification, especially an automated

one.

**traceability:** The ability to identify related items in documentation and software, such as

requirements with associated tests. See also *horizontal traceability*, *vertical traceability*.

**horizontal traceability:** The tracing of requirements for a test level through the layers of test

documentation (e.g. test plan, test design specification, test case specification and test

procedure specification or test script).

**vertical traceability:** The tracing of requirements through the layers of development

documentation to components.

**4.2**

**black box test design technique:** Procedure to derive and/or select test cases based on an

analysis of the specification, either functional or non-functional, of a component or system

without reference to its internal structure.

**experience-based test design technique:** Procedure to derive and/or select test cases based

on the tester’s experience, knowledge and intuition.

**test design technique:** Procedure used to derive and/or select test cases.

**white-box test design technique:** Procedure to derive and/or select test cases based on an

analysis of the internal structure of a component or system.

**4.3**

**boundary value analysis:** A black box test design technique in which test cases are designed

based on boundary values. See also *boundary value.*

**boundary value:** An input value or output value which is on the edge of an equivalence

partition or at the smallest incremental distance on either side of an edge, for example the

minimum or maximum value of a range.

**decision table testing:** A black box test design technique in which test cases are designed to

execute the combinations of inputs and/or stimuli (causes) shown in a decision table.

See also *decision table.*

**decision table:** A table showing combinations of inputs and/or stimuli (causes) with their

associated outputs and/or actions (effects), which can be used to design test cases.

**equivalence partition:** A portion of an input or output domain for which the behavior of a

component or system is assumed to be the same, based on the specification.

**state transition testing:** A black box test design technique in which test cases are designed to

ATA execute valid and invalid state transitions. See also *N-switch testing*.

**N-switch testing:** A form of state transition testing in which test cases are designed to execute

all valid sequences of N+1 transitions.

**use case testing:** A black box test design technique in which test cases are designed to

ATA execute scenarios of use cases.

**4.4**

**code coverage:** An analysis method that determines which parts of the software have been

executed (covered) by the test suite and which parts have not been executed, e.g. statement

coverage, decision coverage or condition coverage.

**decision coverage:** The percentage of decision outcomes that have been exercised by a test

suite. 100% decision coverage implies both 100% branch coverage and 100% statement

coverage.

**statement coverage:** The percentage of executable statements that have been exercised by a

test suite.

**structure-based testing:** See *white-box testing.*

**4.5**

**exploratory testing:** An informal test design technique where the tester actively controls the

design of the tests as those tests are performed and uses information gained while testing to

design new and better tests.

**fault attack:** See *attack.*

**attack:** Directed and focused attempt to evaluate the quality, especially reliability, of a test

object by attempting to force specific failures to occur. See also *negative testing.*

**negative testing:** Tests aimed at showing that a component or system does not work.

Negative testing is related to the testers’ attitude rather than a specific test approach or test

design technique, e.g. testing with invalid input values or exceptions.

**5.1**

**tester:** A skilled professional who is involved in the testing of a component or system.

**test leader:** See *test manager.*

**test manager:** The person responsible for project management of testing activities and

resources, and evaluation of a test object. The individual who directs, controls, administers,

plans and regulates the evaluation of a test object.

**5.2**

**test approach:** The implementation of the test strategy for a specific project. It typically

includes the decisions made that follow based on the (test) project’s goal and the risk

assessment carried out, starting points regarding the test process, the test design techniques

to be applied, exit criteria and test types to be performed

**test strategy:** A high-level description of the test levels to be performed and the testing within

those levels for an organization or programme (one or more projects).

**5.3**

**defect density:** The number of defects identified in a component or system divided by the

size of the component or system (expressed in standard measurement terms, e.g. lines-ofcode,

number of classes or function points).

**failure rate:** The ratio of the number of failures of a given category to a given unit of

measure, e.g. failures per unit of time, failures per number of transactions, failures per

number of computer runs.

**test control:** A test management task that deals with developing and applying a set of

corrective actions to get a test project on track when monitoring shows a deviation from

what was planned.

**test monitoring:** A test management task that deals with the activities related to periodically

checking the status of a test project. Reports are prepared that compare the actuals to that

which was planned. See also *test management*.

**test management:** The planning, estimating, monitoring and control of test activities,

typically carried out by a test manager.

**test summary report:** A document summarizing testing activities and results. It also contains

an evaluation of the corresponding test items against exit criteria.

**5.4**

**configuration management:** A discipline applying technical and administrative direction and

surveillance to: identify and document the functional and physical characteristics of a

configuration item, control changes to those characteristics, record and report change

processing and implementation status, and verify compliance with specified requirements.

**version control:** See *configuration control.*

**configuration control:** An element of configuration management, consisting of the

evaluation, co-ordination, approval or disapproval, and implementation of changes to

configuration items after formal establishment of their configuration identification.

**5.5**

**product risk:** A risk directly related to the test object. See also *risk*.

**risk:** A factor that could result in future negative consequences; usually expressed as impact

ATM and likelihood.

**risk-based testing:** An approach to testing to reduce the level of product risks and inform

stakeholders of their status, starting in the initial stages of a project. It involves the

identification of product risks and the use of risk levels to guide the test process.

**5.6**

**incident logging:** Recording the details of any incident that occurred, e.g. during testing.

**incident management:** The process of recognizing, investigating, taking action and disposing

of incidents. It involves logging incidents, classifying them and identifying the impact.

**incident report:** A document reporting on any event that occurred, e.g. during the testing,

which requires investigation.

**6.1**

**configuration management tool:** A tool that provides support for the identification and

control of configuration items, their status over changes and versions, and the release of

baselines consisting of configuration items.

**coverage tool:** A tool that provides objective measures of what structural elements, e.g.

statements, branches have been exercised by a test suite.

**debugging tool:** A tool used by programmers to reproduce failures, investigate the state of

programs and find the corresponding defect. Debuggers enable programmers to execute

programs step by step, to halt a program at any program statement and to set and examine

program variables.

**dynamic analysis tool:** A tool that provides run-time information on the state of the software

code. These tools are most commonly used to identify unassigned pointers, check pointer

arithmetic and to monitor the allocation, use and de-allocation of memory and to flag

memory leaks.

**incident management tool:** A tool that facilitates the recording and status tracking of

incidents. They often have workflow-oriented facilities to track and control the allocation,

correction and re-testing of incidents and provide reporting facilities.

**load testing tool:** A tool to support load testing whereby it can simulate increasing load, e.g.,

numbers of concurrent users and/or transactions within a specified time-period. See also

*performance testing tool.*

**modeling tool:** A tool that supports the creation, amendment and verification of models of the

software or system

**monitoring tool:** See *monitor.*

**monitor:** A software tool or hardware device that runs concurrently with the component or

system under test and supervises, records and/or analyses the behavior of the component or

system.

**performance testing tool:** A tool to support performance testing that usually has two main

ATT facilities: load generation and test transaction measurement. Load generation can simulate

either multiple users or high volumes of input data. During execution, response time

measurements are taken from selected transactions and these are logged. Performance

testing tools normally provide reports based on test logs and graphs of load against

response times.

**probe effect:** The effect on the component or system by the measurement instrument when

the component or system is being measured, e.g. by a performance testing tool or monitor.

For example performance may be slightly worse when performance testing tools are being

used.

**test comparator:** A test tool to perform automated test comparison of actual results with

expected results.

**test data preparation tool:** A type of test tool that enables data to be selected from existing

databases or created, generated, manipulated and edited for use in testing.

**test design tool:** A tool that supports the test design activity by generating test inputs from a

ATA specification that may be held in a CASE tool repository, e.g. requirements management

tool, from specified test conditions held in the tool itself, or from code.

**test harness:** A test environment comprised of stubs and drivers needed to execute a test.

**test execution tool:** A type of test tool that is able to execute other software using an

automated test script, e.g. capture/playback.

**test management tool:** A tool that provides support to the test management and control part

of a test process. It often has several capabilities, such as testware management, scheduling

of tests, the logging of results, progress tracking, incident management and test reporting.

**unit test framework:** A tool that provides an environment for unit or component testing in

F-AT which a component can be tested in isolation or with suitable stubs and drivers. It also

provides other support for the developer, such as debugging capabilities.

**6.2**

**data-driven testing:** A scripting technique that stores test input and expected results in a

ATT table or spreadsheet, so that a single control script can execute all of the tests in the table.

Data-driven testing is often used to support the application of test execution tools such as

capture/playback tools.

**keyword-driven testing:** A scripting technique that uses data files to contain not only test

data and expected results, but also keywords related to the application being tested. The

keywords are interpreted by special supporting scripts that are called by the control script

for the test. See also *data-driven testing*.

**scripting language:** A programming language in which executable test scripts are written,

used by a test execution tool (e.g. a capture/playback tool).